On the Completeness of Full-Text Search Languages for XML

Chavdar Botev
Cornell University
cbotev@cs.cornell.edu

Sihem Amer-Yahia
AT&T Labs–Research
sihem@research.att.com

Jayavel Shanmugasundaram
Cornell University
jai@cs.cornell.edu

Abstract

We study formal properties of full-text search languages for XML. Our main contribution is the development of a formal model for full-text search based on the positions of tokens in XML nodes. Building on this model, we define a full-text calculus based on first-order logic, and a full-text algebra based on the relational algebra. We show that the full-text calculus and algebra are equivalent even in the presence of arbitrary position-based predicates, such as distance predicates and phrase matching. This suggests a notion of completeness for full-text languages. None of the full-text search languages that we are aware of are complete under the above characterization. We propose a new full-text language that is complete and naturally generalizes existing full-text languages. Our formalization in terms of the relational model can also serve as the basis for (a) the joint optimization of structured and full-text search queries, and (b) ranking full-text search query results by leveraging existing work on probabilistic relational models.

1 Introduction

The emergence of XML has resulted in the rapid growth of semi-structured XML repositories. Examples of such publicly available repositories are the Library of Congress (LoC) documents [15], the IEEE INEX data collection [14], Shakespeare's plays, DBLP and SIGMOD Record. A common characteristic of these repositories is that they have a mix of structured and unstructured data. For example, the LoC XML documents describe congressional bills and contain structured information such as bill date and sponsors and unstructured text such as bill description. Querying the content of such documents requires powerful full-text search primitives, in addition to structured query operations.

While XML query languages such as XPath and XQuery [20] support sophisticated structured query operators such as selections, joins and aggregations, they can only support very rudimentary full-text search. For instance, full-text search in XQuery is expressed using the function: `contains($e, tokens)` which returns true iff the XML node bound to the variable `$e` contains all the tokens in `tokens`. While this function is sufficient for simple sub-string matching, it is woefully inadequate for more complex searches. As an illustration, consider the following example from the XQuery Full-Text Use Cases Document [21].

Example 1 (Use Case 10.4): *Consider an XML document that contains book and article elements. Find the book elements that contain the keywords "efficient" and the phrase "task completion" in that order with at most 10 intervening tokens.*

The above query includes phrase matching, order specifications, and distance predicates. The `contains` function in XQuery cannot express and compose all these full-text primitives. XML full-text languages that we are aware of are also unable to express the above query as well as other full-text operations specified in the XQuery Full-Text Use Cases Document, such as general Boolean conditions and paragraph scoping. We believe that one reason for the current state of affairs is that there is no adequate formal model for full-text

search in XML. This makes it hard to understand the full scope of full-text search in XML and develop appropriate languages that can be tightly integrated with structured XML query languages.

One of the main contributions of this paper is the development of a formal model for full-text search in XML. Our model is based on the notion of positions of tokens within an XML node. Explicitly modeling the positions in an XML node, instead of simply treating nodes as a "bag of words" as in traditional information retrieval (IR), allows us to formally capture position-based predicates such as distance and ordered predicates, in addition to traditional Boolean operators. Building on the above formal model, we define a full-text calculus based on first-order logic, and a full-text algebra based on the relational algebra. The calculus and algebra are powerful enough to model arbitrary position-based predicates. We also show that for any given set of position-based predicates, the full-text calculus and algebra have equivalent expressive power. This suggests a notion of completeness for full-text search languages.

Given the above notion of completeness, we show the incompleteness of existing full-text search languages that we are aware of. We also describe a new full-text search language based on the full-text calculus, which is complete and naturally generalizes existing Boolean full-text search languages. This new language is the formal basis for the TeXQuery full-text language proposal [1], which we have submitted to the W3C Full-Text Task Force, whose charter is to extend XQuery with full-text search capabilities.

Although not discussed in detail in this paper, our full-text algebra based on the relational model can serve (a) as the foundation for the joint optimization of structured and full-text search queries, and (b) as the basis for scoring and ranking full-text search results based on the probabilistic relational model developed for information retrieval [11].

Finally, although our formalism is used to model full-text search in XML documents, our results also apply to "flat" documents, where we simply consider each flat document to be an XML node with no internal structure. However, one of the main benefits of formalizing full-text search for XML is that it provides the formal foundation for tightly integrating full-text search with structured queries.

2 Modeling Full-Text Search in XML

At its core, a full-text search specification for XML has two components: the *search context*, which specifies the set of XML nodes (i.e., the *context nodes*) over which the full-text search is to be performed and, the *full-text condition*, which specifies the condition that should be evaluated on each context node. Only the context nodes that satisfy the condition qualify as answers. In Example 1 in the introduction, the search context is the set of book elements (and not article elements) that satisfy the full-text condition: *contains the keyword "efficient" and the phrase "task completion" in that order with at most 10 intervening tokens.*

In order to specify a full-text search query over XML documents, we need (1) a language to specify the context nodes and, (2) a language to specify the full-text search condition (the full-text search language). For (1), we can use a structured XML query language such as XQuery, which is relationally complete and has a well-defined formal semantics [20]. We thus focus on the full-text search language in this paper..

One important issue in full-text search is scoring and ranking the context nodes based on how well they satisfy the full-text search condition. While we do not explicitly consider specific scoring schemes in our formalism, in Section 3.3, we shall briefly describe how a scoring scheme based on the probabilistic relational model naturally fits into our full-text algebra formalism.

2.1 Modeling Context Nodes

To reason about full-text search languages, we need a formal model for the context nodes. A context node can be any XML node (e.g., element and attribute nodes). Existing models for context nodes are insufficient for expressive full-text search. For instance, the XQuery data model for the book element in Figure 1

```
<book(1) id(2)="1000(3)">
  <author(4)>Elina(5) Rose(6)</author(7)>
  <content(8)>
    <p(9)> Usability(10) of(11) a(12) software(13) measures(14) how(15) well(16)
           the(17) software(18) provides(19) support(20) for(21) quickly(22)
           achieving(23) specified(24) goals(25).
    </p(26)>
  </content(27)>
</book(28)>
```

Figure 1: Positions Example

treats all the text under an element as a single text node (ignore the numbers in parentheses for now). This model is enough to identify sub-strings in the text and evaluate queries such as *find author nodes containing 'Elina'*. However, it is insufficient to answer queries such as *find books that contain the tokens 'usability' and 'testing' with at most three intervening tokens*. Traditional IR models solve part (but not all) of this problem by tokenizing the entire document (or equivalently, context node in our setting), and representing each token separately. In the example in Figure 1, the text in the context node would be modeled as the "bag of tokens" $\{book, id, 1000, author, Elina, Rose, \ldots\}$. However, this model still cannot capture the distance between tokens (we note that some IR languages do, however, support restricted forms of distance predicates; see Section 4.2 for a more detailed comparison with such languages).

In this paper, we explicitly model the *position* of a token in a context node. We argue that this model, although simple, is powerful enough to capture the semantics of existing full-text search languages. Further, it serves as the formal basis for defining position-based predicates such as proximity distance and order predicates. In Figure 1, we have used a simple numeric position (within parenthesis) for each token.

It is important to note that our proposed model does not dictate any specific implementation of positions (such as numeric positions, as shown above). More expressive positions that capture the notions of lines, sentences and paragraphs can be used, and this will enable more sophisticated predicates on positions than simple distance (such as whether two search tokens appear in the same line or sentence). Our language formalisms are extensible with respect to the set of predicates, and more complex position identifiers will enable the use of more complex predicates.

2.2 Model Definition

We now define our formal model. $\mathcal{N}$ is the set of context nodes, $\mathcal{P}$ is the set of positions, and $\mathcal{T}$ is the (possibly infinite) set of tokens. The function $Positions : \mathcal{N} \rightarrow 2^{\mathcal{P}}$ maps a context node to the set of positions in the context node. The function $Token : \mathcal{P} \rightarrow \mathcal{T}$ maps each position to the token stored at that position. In the example in Figure 1, if the context node is denoted by n, then $Positions(n) = \{1, 2, ..., 28\}$, $Token(1) = book$, $Token(2) = id$, and so on.

3 Full-Text Calculus and Algebra

We define a calculus for full-text search based on first-order logic that captures the key notions of search context, positions, and position-based predicates, which were introduced in the previous section. We then define a relation-based full-text algebra that is equivalent to the full-text calculus. We also outline the potential benefits of our formalism.

3.1 Full-Text Calculus

The full-text calculus defines the following predicates to model basic full-text primitives.

- $SearchContext(node)$ is true iff $node \in \mathcal{N}$ (recall that $\mathcal{N}$ is the set of context nodes).
- $hasPos(node, pos)$ is true iff $pos \in Positions(node)$. This predicate explicitly captures the notion of positions in an XML node.
- $hasToken(pos, tok)$ is true iff $tok = Token(pos)$. This predicate captures the relationship between tokens and the positions in which they occur.

A full-text language may also wish to specify an additional set of position-based predicates, $Preds$, depending on user needs. The calculus is general enough to support arbitrary position-based predicates. Specifically, given a set $VarPos$ of position variables, and a set $Consts$ of constants, the calculus can support any predicate of the form: $pred(p_1, ..., p_m, c_1, ..., c_r)$, where $p_1, ...p_m \in VarPos$ and $c_1, ..., c_r \in Consts$. For example, we could define $Preds = \{distance(pos_1, pos_2, dist), ordered(pos_1, pos_2), samepara(pos_1, pos_2), diffpos(pos_1, pos_2)\}$. Here, $distance(pos_1, pos_2, dist)$ returns true iff there are at most $dist$ intervening tokens between pos_1 and pos_2; $ordered(pos_1, pos_2)$ is true iff pos_1 occurs before pos_2; $samepara(pos_1, pos_2)$ is true iff pos_1 is in the same paragraph as pos_2; $diffpos(pos_1, pos_2)$ is true iff pos_1 and pos_2 are different positions.

3.1.1 Full-Text Calculus Queries

A full-text calculus query is of the form: $\{node|SearchContext(node) \wedge QueryExpr(node)\}$. Intuitively, the query returns $node$s that are in the search context, and that satisfy $QueryExpr(node)$. $QueryExpr(node)$, hereafter called the *query expression*, is a first-order logic expression that specifies the full-text search condition. $node$ is the only free variable in the query expression. The structure of the query expression is recursively defined as follows.

- $hasPos(node, pos_i)$ is a query expression where $node$ is the free variable and $pos_i \in VarPos$.
- $hasToken(pos_i, tok)$ is a query expression, where $pos_i \in VarPos$ and $tok \in Consts$.
- $pred(pos_1, ..., pos_m, c_1, ..., c_r)$ is a query expression, where $pred \in Preds$, $pos_i \in VarPos$ and $c_j \in Consts$.
- If qe_1 and qe_2 are query expressions, $\neg qe_1$, $qe_1 \wedge qe_2$, and $qe_1 \vee qe_2$ are query expressions.
- If qe is a query expression, then $\exists pos_i(hasPos(node, pos_i) \wedge qe)$, and $\forall pos_i(hasPos(node, pos_i) \Rightarrow qe)$ are query expressions, where $pos_i \in VarPos$.

A full-text calculus query has the conventional semantics of first-order logic. The form of the quantification in the last bullet guarantees that the query expression in the full-text calculus can be evaluated using only the positions and tokens in the context node, without having to look at other positions. This notion is similar to the notion of safety for the relational calculus.

We now provide some examples of full-text calculus queries. The following query returns the context nodes that contain the tokens 'test' and 'usability'.

$\{node|SearchContext(node) \wedge \exists pos_1(hasPos(node, pos_1) \wedge hasToken(pos_1, 'test') \wedge$
$\exists pos_2(hasPos(node, pos_2) \wedge hasToken(pos_2, 'usability')))\}$

In the subsequent examples, we only show the query expression since the rest of the query is the same. The following query returns the context nodes that contain the token 'test' and the token 'usability' with at most 5 intervening tokens.

$\exists pos_1(hasPos(node, pos_1) \wedge hasToken(pos_1, 'test') \wedge \exists pos_2(hasPos(node, pos_2) \wedge$

$hasToken(pos_2,' usability') \wedge distance(pos_1, pos_2, 5)))$

The following query returns the context nodes that contain two occurrences of the token 'test' and do not contain the token 'usability'.

$\exists pos_1(hasPos(node, pos_1) \wedge hasToken(pos_1,' test') \wedge \exists pos_2(hasPos(node, pos_2) \wedge hasToken(pos_2,' test')$
$\wedge diffpos(pos_1, pos_2) \wedge \forall pos_3(hasPos(node, pos_3) \Rightarrow \neg hasToken(pos_3,' usability'))))$

3.2 Full-Text Algebra

We now define our full-text relations and algebra operators. The underlying data model for our algebra is a *full-text relation* of the form $\texttt{R}[\texttt{CNode}, \texttt{att}_1, ..., \texttt{att}_\texttt{m}]$ where the domain of CNode is $\mathcal{N}$ (context nodes), and the domain of $\texttt{att}_\texttt{i}$ is $\mathcal{P}$ (positions). R satisfies the following properties.

- R has always at least the attribute CNode. This captures the context node for full-text search. The remaining attributes in R capture the essence of full-text search, which is to manipulate positions.
- Each tuple t in a full-text relation should satisfy the condition that for all the positions pos in t, $pos \in Positions(t.CNode)$. The intuition is that a full-text search query can only manipulate positions within a single context node.

A full-text algebra expression is based on the following full-text relations that characterize the search context nodes, their positions, and the tokens at these positions.

- $\texttt{SearchContext}(\texttt{CNode})$: This relation contains a tuple ($node$) for each $node \in \mathcal{N}$.
- $\texttt{HasPos}(\texttt{CNode}, \texttt{att}_1)$: This relation contains a tuple for each ($node$,pos) pair that satisfies: $node \in \mathcal{N} \wedge pos \in Positions(node)$. Intuitively, this relation relates context nodes to their positions.
- $\texttt{R}_\texttt{token}(\texttt{CNode}, \texttt{att}_1)$: This is a family of relations, one for each $token \in \mathcal{T}$. R_{token} contains a tuple for each ($node$,pos) pair that satisfies: $node \in \mathcal{N} \wedge pos \in Positions(node) \wedge token = Token(pos)$. Intuitively, R_{token} contains positions that contain $token$, and is similar to an inverted list in IR.

We note that while each $\texttt{R}_\texttt{token}$ relation is finite, there number of such relations will be infinite if $\mathcal{T}$ is infinite. However, this does not lead to a problem in defining the algebra because each algebra expression is finite, and can only refer to a finite set of such relations. Also, physically instantiating the potentially infinite set of $\texttt{R}_\texttt{token}$ relations is not a problem because only a finite sub-set of these relations will be non-empty (because the search context is finite), so only this finite set of relations will have to be explicitly stored. This is in fact what happens in current implementations of inverted lists.

In addition, as in the calculus, we have a set of position-based predicates $Preds$.

3.2.1 Full-Text Algebra Operators and Queries

The full-text algebra operators are similar to the relational operators, but with two important differences. First, full-text algebra operators only operate on full-text relations (as defined above), and not on arbitrary relations, due to the nature of full-text search. Second, full-text algebra operators implicitly enforce that each operation only manipulates positions within a single node, and not across nodes. These two properties ensure that the full-text algebra is equivalent to the full-text calculus in characterizing full-text search. A full-text algebra expression is defined recursively as follows.

- SearchContext is an algebra expression that returns the tuples in the full-text relation SearchContext.

- HasPos is an algebra expression that returns the tuples in the full-text relation HasPos.
- $\mathtt{R_{token}}$ is an algebra expression that returns the tuples in the $\mathtt{R_{token}}$ relation, where $token \in \mathcal{T}$.
- If $Expr_1$ is an algebra expression, $\pi_{\mathtt{CNode,att_1,...,att_i}}(Expr_1)$ is an algebra expression. If $Expr_1$ evaluates to the full-text relation $\mathtt{R_1}$, the full-text relation corresponding to the new expression is: $\pi_{\mathtt{CNode,att_1,...,att_i}}(\mathtt{R_1})$, where π is the traditional relational projection operator. The attribute names of the result full-text relation are renamed to have consecutive $\mathtt{att_i}$'s. Note that π *always* has to include CNode in the full-text algebra - this enforces the property that full-text search is always scoped within a single context node.
- If $Expr_1$ and $Expr_2$ are algebra expressions, then $(Expr_1 \bowtie Expr_2)$ is an algebra expression, If $Expr_1$ and $Expr_2$ evaluate to R_1 and R_2 repectively, then the full-text relation corresponding to the new expression is: $\mathtt{R_1} \bowtie_{\mathtt{R_1.CNode=R_2.CNode}} \mathtt{R_2}$, where $\bowtie_{\mathtt{R_1.CNode=R_2.CNode}}$ is the traditional relational equi-join operation on the CNode attribute. The duplicate CNode attribute is projected out in the result full-text relation, and the position attributes are renamed to be consecutive $\mathtt{att_i}$'s. Note again how the full-text algebra does not allow operations across nodes because the only predicate that is permitted in the join is equality between the attributes CNode of the two relations.
- If $Expr_1$ is an algebra expression, then $\sigma_{pred(\mathtt{att_1,...,att_m,c_1,...,c_q})}(Expr_1)$ is an algebra expression, where $pred \in Preds$. If $Expr_1$ evaluates to R_1, the full-text relation corresponding to the new expression is: $\sigma_{pred(\mathtt{att_1,...,att_m,c_1,...,c_q})}(\mathtt{R_1})$, where σ is the traditional relational selection operator.
- If $Expr_1$ and $Expr_2$ are algebra expression, then $(Expr_1 - Expr_2)$, $Expr_1 \cup Expr_2$, and $Expr_1 \cap Expr_2$ are algebra expressions. These $-$, $\cup$ and $\cap$ operators have the same semantics as in traditional relational algebra.

A full-text algebra query is a full-text algebra expression that produces a full-text relation with a single attribute (this attribute has to be CNode by definition). The set of nodes in the result full-text relation defines the result of a full-text algebra query.

We now provide some examples of full-text algebra queries that correspond to the calculus example in Section 3.1.1. The following query returns the context nodes that contain the token 'test' and 'usability': $\pi_{\mathtt{CNode}}(\mathtt{R_{test}} \bowtie \mathtt{R_{usability}})$

The following query returns the context nodes that contain the token 'test' and the token 'usability' within a distance of 5: $\pi_{\mathtt{CNode}}(\sigma_{distance(p_1,p_2,5)}(\mathtt{R_{test}} \bowtie \mathtt{R_{usability}}))$

The following query returns the context nodes that contain two occurrences of the token 'test' and do not contain the token 'usability': $\pi_{\mathtt{CNode}}((\sigma_{diffpos(att_1,att_2)}(\mathtt{R_{test}} \bowtie \mathtt{R_{test}})) \bowtie (\mathtt{SearchContext} - \pi_{\mathtt{CNode}}(\mathtt{R_{usability}})))$

3.3 Equivalence of Calculus and Algebra and Its Applications

Theorem 1 *Given a set of position-based predicates $Preds$, the full-text calculus and the full-text algebra are equivalent in expressive power.*

The proof is in Appendix A, and is similar to the equivalence proof for the relational calculus and algebra.

The equivalence of the full-text calculus and algebra suggests a notion of completeness for full-text search languages. This provides a formal basis for comparing the expressive power of different query languages, as we shall do in the next section. To the best of our knowledge, this is the first attempt to formalize the expressive power of full-text search languages, either for flat documents or for XML documents. Developing

a full-text algebra in terms of relations also provides a foundation for tightly integrating, optimizing and evaluating structured (relational or XML) queries with full-text search.

The full-text algebra also enables us to rank query results by leveraging existing work on the probabalistic relational model developed in the context of IR [11, 23]. Specifically, the probabilistic relational model includes a probability attribute for each tuple that specifies its relevance to the result relation. A tuple with a high probability is very relevant to the result relation, while a tuple with low probability is not. In addition, the model defines how these probabilities are propagated through traditional relational operators. In our context, we simply need to add a new probability attribute to our full-text relations. We can then rely on these techniques to propagate this attribute through the algebra operators, and produce ranked results.

4 Incompleteness of Existing Languages and a Complete Language

In this section, we show the incompleteness of existing full-text languages with respect to the algebra and calculus. We then define a complete full-text language based on the full-text calculus that naturally generalizes existing languages.

4.1 Incompleteness of Boolean Full-Text Search Languages

Boolean full-text search languages are commonly used in IR, and have also been proposed for XML full-text search [10, 19]. A typical syntax for such languages, which we shall call `BOOL`, is given below. The simplest query is a search token, which can either be a string literal (such as 'test') or the keyword `ANY`, which matches any token in a node. In addition, the query can be composed with Boolean operators.

Query := Token | `NOT` Query | Query `AND` Query | Query `OR` Query

Token := StringLiteral | `ANY`

We can recursively define the semantics of `BOOL` in terms of our calculus. If the query is a StringLiteral `'token'`, it is equivalent to the calculus query expression $\exists p(hasPos(n,p) \wedge hasToken(p,'token'))$. If the query is `ANY`, it is equivalent to the expression $\exists p(hasPos(n,p))$. If the query is of the form `NOT Query`, it is equivalent to $\neg Expr$, where $Expr$ is the calculus expression for `Query`. If the query is of the form `Query1 AND Query2`, it is equivalent to $Expr1 \wedge Expr2$, where $Expr1$ and $Expr2$ are calculus expressions for `Query1` and `Query2` respectively. `OR` is defined similarly. As an example, the query `'test' AND NOT 'usability'` is equivalent to the calculus query expression: $\exists p_1(hasPos(n,p_1) \wedge hasToken(p_1,'test')) \wedge \neg(\exists p_2 hasPos(n,p_2) \wedge hasToken(p_2,'usability'))$.

Obviously, `BOOL` cannot express position-based predicates. However, we now show that even if we disallow such predicates in the calculus (i.e., $Preds = \phi$), `BOOL` is still incomplete if $\mathcal{T}$ is infinite.

Theorem 2 *If $\mathcal{T}$ is infinite, there exists a full-text query that can be expressed in the full-text calculus with $Preds = \phi$, but which cannot be expressed by* `BOOL`.

Proof Sketch: We shall show that no query in `BOOL` can express the following calculus query: $\exists p(hasPos(n,p) \wedge \neg hasToken(p,t_1))$ (i.e., *find context nodes that contain at least one token that is not* t_1), where $t_1 \in \mathcal{T}$. The proof is by contradiction. Assume that there exists a query $\mathcal{Q}$ in `BOOL` that can express the calculus query. Let $\mathcal{T}_\mathcal{Q}$ be the set of tokens that appear in $\mathcal{Q}$. We construct two context nodes CN_1 and CN_2. CN_1 contains only the token t_1. CN_2 contains the token t_1 and one other token $t_2 \in \mathcal{T} - (\mathcal{T}_\mathcal{Q} \cup \{t_1\})$ (such a token t_2 always exists because $\mathcal{T}$ is infinite and $\mathcal{Q}$ is finite). By the construction, we can see that CN_1 does not satisfy the calculus query, while CN_2 does. We will now show that $\mathcal{Q}$ either returns both CN_1 or CN_2 or neither of them; since this contradicts our assumption, this will prove the theorem.

Let C_Q be the calculus expression equivalent to $\mathcal{Q}$. We show by induction on the structure of C_Q that every sub-expression of C_Q (and hence C_Q) returns the same Boolean value for CN_1 and CN_2. If the

sub-expression is of the form $\exists p(hasPos(n,p) \wedge hasToken(p,token))$, it returns true for both CN_1 and CN_2 if $token = t_1$, and false if $token \neq t_1$ (by construction of CN_1 and CN_2 - recall that $token$ appears in $\mathcal{Q}$). If the sub-expression is of the form $\exists p(hasPos(n,p))$, it returns true for both CN_1 and CN_2. If the sub-expression is of the form $\neg Expr$, then it returns the same Boolean value for both CN_1 and CN_2 because $Expr$ returns the same Boolean value (by induction). A similar argument can also be made for the $\wedge$ and $\vee$ Boolean operators. □

If we limit $\mathcal{T}$ to be finite, however, we can prove that BOOL is complete with $Preds = \phi$.

Theorem 3 *If $\mathcal{T}$ is finite, every query that can be expressed in the full-text calculus with $Preds = \phi$ can be expressed in* BOOL.

The proof is presented in Appendix A. The main intuition is that, if $\mathcal{T}$ is finite, we can express queries such as: $\exists p(hasPos(n,p) \wedge \neg hasToken(p,t_1))$ in BOOL by explicitly listing all the tokens that are not t_1. Although BOOL is complete under this assumption, it is not always practical because even for simple queries such as the one above, we need to explicitly list all possible tokens other than t_1 in the query.

4.2 Incompleteness of Existing Predicate-Based Full-Text Search Languages

We now consider full-text languages that have position-based predicates in addition to Boolean operators [2, 4]. A typical syntax for such a language, which we shall call DIST, is given below.

Query := Token | NOT Query | Query AND Query | Query OR Query | dist(Token,Token,Integer)

Token := StringLiteral | ANY

The semantics of DIST is the same as BOOL, except for the addition of `dist(Token,Token,Integer)`. This construct is the equivalent of the $distance$ predicate introduced in the calculus (Section 3.1), and specifies that the number of intervening tokens should be less than the specified integer. More formally, the semantics of dist(t_1,t_2,d) for some tokens t_1 and t_2 and some integer d is given by the calculus expression: $\exists p_1(hasPos(n,p_1) \wedge hasToken(p_1,t_1) \wedge \exists p_2(hasPos(n,p_2) \wedge hasToken(p_2,t_2) \wedge distance(p_1,p_2,d)))$. If t_1 or t_2 is ANY instead of a string literal, then the corresponding $hasToken$ predicate is omitted in the semantics. We now show that DIST is incomplete with respect to the calculus so long as $\mathcal{T}$ is not trivially small. We can also prove similar incompleteness results for other position-based predicates.

Theorem 4 *If $\mid \mathcal{T} \mid \geq 2$, there exists a full-text query that can be expressed in the full-text calculus with $Preds = \{distance(p_1,p_2,d)\}$, but which cannot be expressed by* DIST.

Proof Sketch: We shall show that no query in DIST can express the following calculus query: $\exists p_1(hasPos(n,p_1) \wedge \exists p_2(hasPos(n,p_2) \wedge hasToken(p_1,t_1) \wedge hasToken(p_2,t_2) \wedge \neg distance(p_1,p_2,0)))$, where $t_1 \in \mathcal{T}$, $t_2 \in \mathcal{T}$ and $t_1 \neq t_2$ (i.e., *find context nodes where the tokens t_1 and t_2 do not appear next to each other at least once*). The proof is by contradiction. Assume that there exists a query $\mathcal{Q}$ in DIST that can express the calculus query. We now construct two context nodes CN_1 and CN_2 as follows. CN_1 contains the tokens t_1 followed by t_2 followed by t_1. CN_2 contains the tokens t_1 followed by t_2 followed by t_1 followed by t_2. By the construction, we can see that CN_1 does not satisfy the calculus query, while CN_2 does. Using induction on the structure of $\mathcal{Q}$ similar to the proof of Theorem 2, we can show that $\mathcal{Q}$ either returns both CN_1 or CN_2 or neither of them. This is a contradiction. □

4.3 A Complete Full-Text Query Language

We now present a new language COMP based on the full-text calculus that is complete even in the presence of arbitrary position-based predicates. COMP shares the same syntax as BOOL for simple Boolean queries,

but naturally generalizes BOOL with position variables to achieve completeness. Thus, simple queries retain the same conventional syntax, while new constructs are only required for more complex queries.

Query := Token | NOT Query | Query AND Query | Query OR Query | SOME Var Query | EVERY Var Query | Preds
Token := StringLiteral | ANY | Var HAS StringLiteral | Var HAS ANY
Preds := distance(Var,Var,Integer) | diffpos(Var,Var) | ...

The main additions to BOOL are the HAS construct in Token, and the SOME, EVERY and Preds constructs in Query (the semantics of the other constructs remain unchanged from BOOL). The HAS construct allows us to explicitly bind position variables (Var) to positions where tokens occur. The semantics for 'var_1 HAS tok' in terms of the calculus, where tok is a StringLiteral is: $hasToken(var_1, tok)$. The semantics for 'var_1 HAS ANY' is: $hasPos(n, var_1)$. While the HAS construct allows us to explicitly bind position variables to token positions, the SOME and EVERY constructs allows us to quantify over these positions. The semantics of 'SOME var_1 Query' is $\exists var_1(hasPos(n, var_1) \wedge Expr)$, where $Expr$ is the calculus expression semantics for Query. The semantics of 'EVERY var_1 Query' is $\forall var_1(hasPos(n, var_1) \Rightarrow Expr)$, where $Expr$ is the calculus expression semantics for Query. Finally, the Preds construct allows for the definition of arbitrary position-based predicates. The semantics of a predicate 'pred(var_1,...,var_p,c_1,...c_q)', is simply: $pred(var_1, \ldots, var_p, c_1, \ldots, c_q)$.

As an illustration of the power of COMP, the following two queries express the calculus queries used to prove the incompleteness of BOOL and DIST in Theorems 2 and 4, respectively.

SOME p_1 (NOT p_1 HAS t_1)

SOME p_1 SOME p_2 (p_1 HAS t_1 AND p_2 HAS t_2 AND NOT distance(p_1,p_2,0))

We can prove that COMP is complete (the proof is in the appendix).

Theorem 5 *Every query that can be expressed in the full-text calculus using predicates $Preds$ can be expressed by* COMP *using $Preds$.*

5 Related Work

Most of IR research [2][18] has focused on methods for relevance estimation and efficient evaluation of keyword queries. In this context, full-text languages have been developed to implement specific primitives, but their formal properties such as expressive power and completeness have not been studied. This observation also applies to XML full-text search languages such as XQuery/IR [3], XIRQL [10], XSEarch [8], XRANK [12], XXL [19] and Niagara [22]. Existing work on probabilistic relational databases [11, 23] could be used to extend our algebra to support relevance scoring of query results. Several other works have used relational systems to store inverted lists and translate keyword queries to SQL [5, 9, 13, 16, 17, 22]. Such implementations do not provide a formal basis for completeness and could benefit from our formalization. Finally, our study of completeness for full-text languages is similar to the that for the relational algebra and calculus [7], and goal is to provide a similar formal basis for full-text querying so that it can be tightly integrated with structured queries.

6 Conclusion

This paper presents a simple, yet powerful formalization of full-text search for XML. While this paper makes an initial step in characterizing full-text query languages for XML, there are multiple directions we can explore to build on this work. First, we wish to explore how the formal characterization of full-text search in terms of the relational model can enable the seamless integration of full-text languages with structured query languages, in terms of both query optimization and query evaluation. Second, we wish

to incorporate primitives such as stemming, thesaurus and stop-words in our framework. We believe that quantifi cation over tokens adds additional expressive power, and would enable these additional features.

References

[1] S. Amer-Yahia, C. Botev, J. Robie, J. Shanmugasundaram. TeXQuery: A Full-Text Search Extension to XQuery. http:/www.cs.cornell.edu/database/TeXQuery/.

[2] R. Baeza-Yates, B. Ribiero-Neto. Modern Information Retrieval. Addison-Wesley, 1999.

[3] J. M. Bremer, M. Gertz. XQuery/IR: Integrating XML Document and Data Retrieval. WebDB 2002.

[4] E. W. Brown. Fast Evaluation of Structured Queries for Information Retrieval. SIGIR 1995.

[5] T. T. Chinenyanga, N. Kushmerick. Expressive and Efficient Ranked Querying of XML Data. WebDB 2001.

[6] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. Commun. ACM 13(6): 377-387 (1970).

[7] E.F. Codd. Relational Completeness of Database Sublanguages. In R. Rustin (ed.), Database Systems, Prentice-Hall, 1972.

[8] S. Cohen et al. XSEarch: A Semantic Search Engine for XML. VLDB 2003.

[9] D. Florescu, D. Kossmann, I. Manolescu. Integrating Keyword Search into XML Query Processing. WWW 2000.

[10] N. Fuhr, K. Grossjohann. XIRQL: An Extension of XQL for Information Retrieval. SIGIR 2000.

[11] N. Fuhr, T. Rölleke. A Probabilistic Relational Algebra for the Integration of Information Retrieval and Database Systems. ACM TOIS 15(1), 1997.

[12] L. Guo, F. Shao, C. Botev, J. Shanmugasundaram. XRANK: Ranked Keyword Search over XML Documents. SIGMOD 2003.

[13] V. Hristidis, L. Gravano, Y. Papakonstantinou. Efficient IR-Style Keyword Search over Relational Databases. VLDB 2003.

[14] Initiative for the Evaluation of XML Retrieval. http://www.is.informatik.uni-duisburg.de/projects/inex03/.

[15] Library of Congress. http://lcweb.loc.gov/crsinfo/xml/.

[16] J. Melton, A. Eisenberg. SQL Multimedia and Application Packages (SQL/MM). SIGMOD Record 30(4), 2001.

[17] A. Salminen. A Relational Model for Unstructured Documents. SIGIR 1987.

[18] G. Salton, M. J. McGill. Introduction to Modern Information Retrieval. McGraw-Hill, 1983.

[19] A. Theobald, G. Weikum. The Index-Based XXL Search Engine for Querying XML Data with Relevance Ranking. EDBT 2002.

[20] The World Wide Web Consortium. XQuery 1.0: An XML Query Language. W3C Working Draft. http://www.w3.org/TR/xquery/.

[21] The World Wide Web Consortium. XQuery and XPath Full-Text Use Cases. W3C Working Draft. http://www.w3.org/TR/xmlquery-full-text-use-cases/.

[22] C. Zhang, J. Naughton, D. DeWitt, Q. Luo, G. Lohman. On Supporting Containment Queries in Relational Database Management Systems. SIGMOD 2001.

[23] E. Zimanyi. Query Evaluations in Probabilistic Relational Databases. Theoretical Computer Science, 1997.

A Proofs of Theorems

Theorem 1: Equivalence of the Full-Text Calculus and Algebra

Lemma 1. For every full-text algebra expression that only uses position-based predicates from the set $Preds$, there exists an equivalent full-text calculus expression that only uses position-based predicates from the same set $Preds$.

Proof Sketch: We will prove that for every algebra expression $AlgExpr$, which evaluates to a relation $R(CNode, att_1, att_2, ..., att_k)$, $k \geq 0$, there exists a calculus query expression $CalcExpr(n, p_1, ..., p_k)$ with free variables $\{n, p_1, ..., p_k\}$, such that $\{(n, p_1, ..., p_k) \mid SearchContext(n) \wedge hasPos(n, p_1) \wedge ... \wedge hasPos(n, p_k) \wedge CalcExpr(n, p_1, ..., p_k)\} = R$.

The proof is by induction on the structure of $AlgExpr$.

- If $AlgExpr = \texttt{SearchContext}$, then $CalcExpr(n) = (\exists p\ hasPos(n, p) \wedge hasPos(n, p)) \vee \neg(\exists p\ hasPos(n, p) \wedge hasPos(n, p))$. $CalcExpr(n)$ is always true; therefore, $\{n \mid SearchContext(n) \wedge CalcExpr(n)\}$ is equal to the full-text relation $SearchContext$ by its definition.

- If $AlgExpr = \texttt{HasPos}$, then $CalcExpr(n, p_1) = hasPos(n, p_1)$, i.e. $\{(n, p_1) \mid SearchContext(n) \wedge hasPos(n, p_1)\}$ is equal to the full-text relation $HasPos$ by its definition.

- If $AlgExpr = \texttt{R}_{\texttt{token}}$, then $CalcExpr(n) = hasToken(p, 'token')$. $\{(n, p_1) \mid SearchContext(n) \wedge hasPos(n, p_1) \wedge hasToken(p_1, 'token')\}$ is equal to the full-text relation R_{token} by its definition.

- If $AlgExpr = \pi_{\texttt{CNode,att}_1,...,\texttt{att}_\texttt{i}}(\texttt{AlgExpr}')$, where $AlgExpr'$ is a full-text algebra expression whose equivalent calculus query expression is $CalcExpr'(n, p_1, ..., p_m)$ and $AlgExpr'$ evaluates to $R'(CNode, att_1, att_2, ..., att_m)$, $m \geq i$, then $CalcExpr(n, p_1, ..., p_i) = \exists p_{i+1}\ hasPos(n, p_{i+1}) \wedge ...\exists p_m\ hasPos(n, p_m) \wedge CalcExpr'(n, p_1, ..., p_m)$. $\{(n, p_1, ..., p_i) \mid SearchContext(n) \wedge \bigwedge_{j=1,...,i} hasPos(n, p_j) \wedge CalcExpr(n, p_1, ..., p_i)\} = \pi_{CNode,p_1,...,p_i}\{(n, p_1, ..., p_m) \mid SearchContext(n) \wedge \bigwedge_{j=1,...,m} hasPos(n, p_j) \wedge CalcExpr'(n, p_1, ..., p_m)\} = \pi_{CNode,p_1,...,p_i}(R')$.

- If $AlgExpr = \texttt{AlgExpr}_1 \bowtie \texttt{AlgExpr}_2$, where $AlgExpr_1$ and $AlgExpr_2$ are full-text algebra expressions that evaluate to $R_i(CNode, att_1, ..., att_{m_i})$ for $i = 1, 2$, and their equivalent calculus query expressions are $CalcExpr_i(n, p_1, ..., p_{m_i})$ for $i = 1, 2$,then $CalcExpr(n, p_1, ..., ..., p_{m_1+m_2}) = CalcExpr(n, p_1, ..., p_{m_1}) \wedge CalcExpr(n, p_{m_1+1}, ..., p_{m_1+m_2})$. $\{(n, p_1, ..., p_{m_1+m_2} \mid SearchContext(n) \wedge \bigwedge_{j=1,...,m_1+m_2} hasPos(n, p_j) \wedge CalcExpr(n, p_1, ..., p_{m_1}) \wedge CalcExpr(n, p_{m_1+1}, ..., p_{m_1+m_2}\} = R_1 \bowtie R_2$.

- If $AlgExpr = \sigma_{\texttt{pred(att}_1,...,\texttt{att}_\texttt{m},\texttt{c}_1,...,\texttt{c}_\texttt{q})}(\texttt{Expr}')$, where $Expr'$ is a full-text algebra expression that evaluates to the relaton R' and the equivalent calculus query expression is $CalcExpr'(n, p_1, ..., p_k)$, then $CalcExpr(n, p_1, ..., p_k) = CalcExpr'(n, p_1, ..., p_k) \wedge pred(att_1, ..., att_m, c_1, ..., c_q)$. $\{(n, p_1, ..., p_k) \mid SearchContext(n) \wedge \bigwedge_{j=1,...,k} hasPos(n, p_j) \wedge CalcExpr'(n, p_1, ..., p_k) \wedge pred(att_1, ..., att_m, c_1, ..., c_q)\} = \sigma_{pred(att_1,...,att_m,c_1,...,c_q)}R'$.

- Let $AlgExpr = \texttt{AlgExpr}_1 \cup \texttt{AlgExpr}_2$, where $AlgExpr_1$ and $AlgExpr_2$ are full-text algebra expressions that evaluate to R_i for $i = 1, 2$ and their equivalent calculus query expressions are $CalcExpr_i(n, p_1, ..., p_k)$ for $i = 1, 2$, then $CalcExpr(n, p_1, ...p_k) = CalcExpr_1(n, p_1, ...p_k) \vee CalcExpr_2(n, p_1, ...p_k)$. $\{(n, p_1, ..., p_k) \mid SearchContext(n) \wedge \bigwedge_{j=1,...,k} hasPos(n, p_j) \wedge (CalcExpr_1(n, p_1, ...p_k) \vee CalcExpr_2(n, p_1, ...p_k))\} = R_1 \cup R_2$.

- Let $AlgExpr = \texttt{AlgExpr}_1 \cap \texttt{AlgExpr}_2$, where $AlgExpr_1$ and $AlgExpr_2$ are full-text algebra expressions that evaluate to R_i for $i = 1, 2$ and their equivalent calculus query expressions are

$CalcExpr_i(n, p_1, ..., p_k)$ for $i = 1, 2$, then $CalcExpr(n, p_1, ...p_k) = CalcExpr_1(n, p_1, ...p_k) \wedge CalcExpr_2(n, p_1, ...p_k)$. $\{(n, p_1, ..., p_k) \mid SearchContext(n) \wedge \bigwedge_{j=1,...,k} hasPos(n, p_j) \wedge (CalcExpr_1(n, p_1, ...p_k) \wedge CalcExpr_2(n, p_1, ...p_k))\} = R_1 \cap R_2$.

- Let $AlgExpr = \texttt{AlgExpr}_1 - \texttt{AlgExpr}_2$, where $AlgExpr_1$ and $AlgExpr_2$ are full-text algebra expressions that evaluate to R_i for $i = 1, 2$ and their equivalent calculus query expressions are $CalcExpr_i(n, p_1, ..., p_k)$ for $i = 1, 2$, then $CalcExpr(n, p_1, ...p_k) = CalcExpr_1(n, p_1, ...p_k) \wedge \neg CalcExpr_2(n, p_1, ...p_k)$. $\{(n, p_1, ..., p_k) \mid SearchContext(n) \wedge \bigwedge_{j=1,...,k} hasPos(n, p_j) \wedge (CalcExpr_1(n, p_1, ...p_k) \wedge \neg CalcExpr_2(n, p_1, ...p_k))\} = R_1 - R_2$.

This completes the structural induction. The requirement that full- text algebra queries evaluate to a relation with a single $CNode$ attribute ensures that the corresponding $CalcExpr$ expression will have only one free variable - n. Therefore, $\{n|SearchContext(n) \wedge CalcExpr(n)\}$ is a valid calculus query. □

Lemma 2. For every full-text calculus expression that only uses position-based predicates from the set $Preds$, there exists an equivalent full-text algebra expression that only uses position-based predicates from the same set $Preds$.

Proof Sketch: We will prove that for every query calculus expression $CalcExpr(n, p_1, ..., p_k)$ with free variables $\{n, p_1, ..., p_k\}$, $k \geq 0$, there exists an algebra expression $AlgExpr$, which evaluates to a relation $R(CNode, att_1, att_2, ..., att_k)$, such that $\{(n, p_1, ..., p_k) \mid SearchContext(n) \wedge \bigwedge_{j=1,..,k} hasPos(n, p_j) \wedge CalcExpr(n, p_1, ..., p_k)\} = R$.

The proof is by induction on the structure of $CalcExpr$.

- If $CalcExpr(n, p) = hasPos(n, p)$, then $AlgExpr = \texttt{HasPos}$. The proof of the equivalence is the same as the analogous case from Lemma 1.

- If $CalcExpr(n, p) = hasToken(p, 'token')$, then $AlgExpr = \texttt{R}_{\texttt{token}}$. The proof of the equivalence is the same as the analogous case from Lemma 1.

- If $CalcExpr(n, p_1, ..., p_k) = pred(p_1, ..., p_k, c_1, ..., c_q)$, then $AlgExpr = \sigma_{\texttt{pred}(\texttt{p}_1,...,\texttt{p}_k,\texttt{c}_1,...,\texttt{c}_q)}$ ($\texttt{HasPos} \bowtie ... \bowtie \texttt{HasPos}$), where the number of joins is k. Obviously, $R = \{(n, p_1, ..., p_k) \mid SearchContext(n) \wedge \bigwedge_{i=1,...,k} hasPos(n, p_i) \wedge pred(p_1, ..., p_k, c_1, ..., c_q)\}$.

- If $CalcExpr(n, p_1, ..., p_l, q'_1, ..., q'_m, q''_1, ..., q''_c) = CalcExpr_1(n, p_1, ..., p_l, q'_1, ..., q'_m) \wedge CalcExpr_2(n, p_1, ..., p_l, q''_1, ..., q''_c)$, where $k = l + m + c$, $CalcExpr_1$, and $CalcExpr_2$ are calculus query expressions with equivalent algebra expressions $AlgExpr_1$ and $AlgExpr_2$, which evaluate to $R_1(CNode, att_1, ..., att_k, att'_1, ..., att'_m)$ and $R_2(CNode, att_1, ..., att_l, att''_1, ..., att''_c)$, then $AlgExpr = (\texttt{AlgExpr}_1 \bowtie \pi_{\texttt{CNode},\texttt{q}''_1,...,\texttt{q}''_\texttt{c}} \texttt{AlgExpr}_2) \cap (\pi_{\texttt{CNode},\texttt{q}'_1,...,\texttt{q}'_\texttt{m}} \texttt{AlgExpr}_1 \bowtie \texttt{AlgExpr}_2)$. $R = \{(n, p_1, ..., p_l, q'_1, ..., q'_m, q''_1, ..., q''_c) \mid (n, p_1, ..., p_l, q'_1, ..., q'_m) \in R_1 \wedge (n, p_1, ..., p_l, q''_1, ..., q''_c) \in R_2\}$ = $\{(n, p_1, ..., p_l, q'_1, ..., q'_m, q''_1, ..., q''_c) \mid CalcExpr_1(n, p_1, ..., p_l, q'_1, ..., q'_m) \wedge CalcExpr_2(n, p_1, ..., p_l, q''_1, ..., q''_c)\}$, which is exactly what we wnated to show.

- If $CalcExpr(n, p_1, ..., p_l, q'_1, ..., q'_m, q''_1, ..., q''_c)$ = $CalcExpr_1(n, p_1, ..., p_l, q'_1, ..., q'_m) \vee CalcExpr_2(n, p_1, ..., p_l, q''_1, ..., q''_c)$, where $k = l + m + c$, $CalcExpr_1$, and $CalcExpr_2$ are calculus query expressions with equivalent algebra expressions $AlgExpr_1$ and $AlgExpr_2$, which evaluate to $R_1(CNode, att_1, ..., att_k, att'_1, ..., att'_m)$ and $R_2(CNode, att_1, ..., att_l, att''_1, ..., att''_c)$, then $AlgExpr = (\texttt{AlgExpr}_1 \bowtie \pi_{\texttt{CNode},\texttt{q}''_1,...,\texttt{q}''_\texttt{c}} \texttt{AlgExpr}_2) \cup (\pi_{\texttt{CNode},\texttt{q}'_1,...,\texttt{q}'_\texttt{m}} \texttt{AlgExpr}_1 \bowtie \texttt{AlgExpr}_2)$. $R = \{(n, p_1, ..., p_l, q'_1, ..., q'_m, q''_1, ..., q''_c) \mid (n, p_1, ..., p_l, q'_1, ..., q'_m) \in R_1 \vee (n, p_1, ..., p_l, q''_1, ..., q''_c) \in R_2\}$ = $\{(n, p_1, ..., p_l, q'_1, ..., q'_m, q''_1, ..., q''_c) \mid CalcExpr_1(n, p_1, ..., p_l, q'_1, ..., q'_m) \vee CalcExpr_2(n, p_1, ..., p_l, q''_1, ..., q''_c)\}$, which is what we wanted to show.

- Let us consirder the case $CalcExpr(n, p_1, ..., p_k) = \neg CalcExpr'(n, p_1, ..., p_k)$, where $CalcExpr'$ is a calculus query expression that is equivalent to the algebra expression $AlgExpr'$, which evaluates to $R'(n, p_1, ..., p_k)$. If $k > 0$, then $AlgExpr = (\texttt{HasPos} \bowtie ... \bowtie \texttt{HasPos}) - AlgExpr'$, where the number of joins is k. $R = \{(n, p_1, ..., p_k) \mid SearchContext(n) \wedge \bigwedge_{i=1,...,k} hasPos(n, p_i) \wedge \neg CalcExpr'(n, p_1, ..., p_k)\}$, which is what we wanted to show.

  If $k = 0$, then $AlgExpr = \texttt{SearchContext} - \texttt{AlgExpr}'$ and $R = \{n \mid SearchContext(n) \wedge SearchContext(n) \wedge \neg CalcExpr'(n)\}$

- If $CalcExpr(n, p_1, ..., p_k) = \exists p_{k+1}\ hasNode(n, p_{k+1}) \wedge CalcExpr'(n, p_1, ..., p_{k+1})$, where $CalcExpr'$ is a calculus query expression that is equivalent to the algebra expression $AlgExpr'$, which evaluates to R', then $AlgExpr = \pi_{\texttt{CNode},\texttt{p}_1,...,\texttt{p}_k}\texttt{R}'$ and $R = \{(n, p_1, ..., p_k) \mid SearchContext(n) \wedge \exists p_{k+1}\ (n, p_1, ..., p_{k+1}) \in R'\} = \{(n, p_1, ..., p_k) \mid SearchContext(n) \wedge \exists p_{k+1}\ CalcExpr'(n, p_1, ..., p_{k+1})\}$.

- Let $CalcExpr(n, p_1, ..., p_k) = \forall p_{k+1}\ hasNode(n, p_{k+1}) \Rightarrow CalcExpr'(n, p_1, ..., p_{k+1})$, where $CalcExpr'$ is a calculus query expression that is equivalent to the algebra expression $AlgExpr'$. We use the equation $CalcExpr(n, p_1, ..., p_k) = \neg\exists p_{k+1}\ \neg CalcExpr'(n, p_1, ..., p_{k+1})$ and apply the previous case..

For every calculus query, its query expression has only one free variable, n, therefore the equivalent algebra query evaluates to a relation that contains a single column, $CNode$. Therfore, it is a valid algebra query. □

The above two Leammas prove the equivalence of the full-text calculus and algebra.

Theorem 3: Completeness of BOOL when $\mathcal{T}$ is finite

Proof Sketch: Let $F = \{n \mid SearchContext(n) \wedge P(n)\}$ be a calculus query expression. We will prove that there exists an equivalent *Query* expression E in BOOL. Without loss of generality, we assume that every quantifi ed variable in F has a unique name. Let these position variable names be $p_1, p_2, ..., p_m$.

We fi rst normalize $P(n)$ using the sequence of equivalence transformations presented below.

1. *(Sink Negations)* Move all negations down to the predicates $hasPos(n, p_i)$ and $hasToken(p_i, t)$. Replace any repetitive negations $\neg\neg A$ with A. Invert quantifi ers: $\neg\exists p\ hasPos(n, p) \wedge A$ is replaced with $\forall p\ hasPos(n, p) \Rightarrow \neg A$ and $\neg\forall p\ hasPos(n, p) \Rightarrow A$ is replaced with $\exists p\ hasPos(n, p) \wedge \neg A$.

2. *(Group)* Move every expression of the form $hasToken(p_i, t)$ and $\neg hasToken(p_i, t)$ out of the scope of any quantifi er over a variable different from p_i. This is possible because $hasToken$ is applied on only one position variable. Formally, the transformation is a repeated application of $\mathcal{Q}p_j\ A \circ B \mapsto B \circ \mathcal{Q}p_j\ A$ where $\mathcal{Q} \in \{\exists, \forall\}$, $\circ \in \{\wedge, \vee\}$, and B has no free variable p_j. Use the commutativity of $\wedge$ and $\vee$ to group the above predicate expressions next to each other and right after $hasPos(n, p_i)$. We get a propositional formula with propositions of the form $\mathcal{Q}_i p_i\ A_i(n, p_i)$ where $\mathcal{Q}_i \in \{\exists, \forall\}$.

3. *(Remove universal quantification)* Remove any universal quantifers by replacing $\forall p_i\ hasPos(n, p_i) \Rightarrow X$ with $\neg\exists p_i\ hasPos(n, p_i) \wedge \neg\ X$. We get a propositional formula over propositions of the form $\exists p_i\ hasPos(n, p_i) \wedge B_i(n, p_i)$.

4. *(Local DNF)* Convert each $B_i(n, p_i)$ to DNF.

5. *(Split)* Replace $\exists p\ hasPos(n, p) \wedge (X(n, p) \vee Y(n, p))$ with $(\exists p'\ hasPos(n, p') \wedge X(n, p')) \vee (\exists p''\ hasPos(n, p'') \wedge Y(n, p''))$ to $\exists p_i\ hasPos(n, p_i) \wedge B_i(n, p_i)$ for every disjunct in $B_i(n, p_i)$. Let the new position variables be $q_1, ..., q_k$. We get a propositional formula over propositions of the form $\exists q_j\ hasPos(n, q_j) \wedge C_j(n, q_j)$ where C_j is a conjunction.

6. *(Global DNF)* Consider F to be a propositional formula over propositions of the form $\exists q_j \ hasPos(n, q_j) \wedge C_j(n, q_j)$. Convert it to a DNF.

We define $QE(F)$ for a calculus query expression F as the equivalent query in BOOL.

We observe that after the normalization $F = \{n \mid SearchContext(n) \wedge (\bigvee_i \bigwedge_j D_{i,j})\} = \bigcup_i \bigcap_j \{n \mid SearchContext(n) \wedge D_{i,j}\}$, where each $D_{i,j}$ is either of the form $\exists q \ hasPos(n, p) \wedge C(n, q)$ or of the form $\neg\exists q \ hasPos(n, p) \wedge C(n, q)$, as in step *GlobalDNF* from the normalization. Therefore, F can be decomposed into the calculus expressions $F_{i_j} = \{n \mid SearchContext(n) \wedge D_{i_j}\}$ and it is not difficult to see that $QE(F)$ = ($QE(F_{1,1})$ `AND` ... `AND` $QE(F_{1,r_1})$) `OR` ... `OR` ($QE(F_{s,1})$ `AND` ... `AND` QE(F_{s,r_s})). Thus, we can focus only on converting each $F_{i,j}$.

As seen above, each $F_{i,j}$ is of the form $\exists p \ hasPos(n, p) \wedge \bigwedge_r H_r(n, p)$ or of the form $\neg\exists p \ hasPos(n, p) \wedge \bigwedge_r H_r(n, p)$, where $H_r(n, p)$ is $hasToken(p, t)$ or $\neg hasToken(p, t)$. In either case, we can consider there are no duplicates among $H_r(n, p)$; otherwise, we can simply eliminate them.

Let us first consider the case where $F_{i,j} = \exists p \ hasPos(n, p) \wedge \bigwedge_t H_t(n, p)$.

- If there exists r_1 and r_2 such that $H_{r_1}(n, p) = hasToken(p, t_1)$, $H_{r_2}(n, p) = hasToken(p, t_2)$, and $t_1 \neq t_2$, then the condition "one token per position" (Section 3.1) is violated. Therefore, $F_{i,j}$ is the empty set. $QE(F_{i,j})$ = `ANY AND NOT(`$t_1 OR ... OR t_c$`)`, where $\mathcal{T} = \{t_1, ..., t_c\}$ is the set of all tokens. Intuitively, this query returns the empty set because it requires the result nodes to contain a token that is not in $\mathcal{T}$, which is impossible.

- If there exists r_1 and r_2 such that $H_{r_1}(n, p) = hasToken(p, t)$ and $H_{r_2}(n, p) = \neg hasToken(p, t)$ then this is an obvious contradiction and $F_{i,j}$ is the empty set. We define $QE(F_{i,j})$ = `ANY AND NOT(`$t_1 OR ... OR t_c$`)` as above.

- Let there exists r_1 and there does not exist r_2 such that $H_{r_1}(n, p) = hasToken(p, t)$ and $H_{r_2}(n, p) = \neg hasToken(p, t)$. Then we can ignore any $H_r(n, p)$ which contains $\neg hasToken(p, t')$ for some token $t' \neq t$. The latter are trivially satisfied. In this case, $F_{i,j} = \{n \mid SearchContext(n) \wedge \exists p \ hasPos(n, p) \wedge hasToken(p, t)\}$, which is exactly the semantics for $QE(F_{i,j})$ = t.

- The last case is $F_{i,j} = \{n \mid SearchContext(n) \wedge \exists p \ hasPos(n, p) \wedge \neg hasToken(p, t_{i_1}) \wedge ... \wedge \neg hasToken(p, t_{i_v})$. This expression can be interpreted as the condition that n contains a token from the complement $\{t_{j_1}, ..., t_{j_u}\}$ of $\{t_{i_1}, ..., t_{i_v}\}$ with regards to the set $\mathcal{T} : \{t_{j_1}, ..., t_{j_u}\} = \mathcal{T} - \{t_{i_1}, ..., t_{i_v}\}$. Due to the finiteness of $\mathcal{T}$, $F_{i,j} = \{n \mid SearchContext(n) \wedge \exists p \ hasPos(n, p) \wedge (hasToken(p, t_{j_1}) \vee ... \vee hasToken(p, t_{j_u}))\}$ = $\bigcup_r \{n \mid SearchContext(n) \wedge \exists p \ hasPos(n, p) \wedge hasToken(p, t_{j_r})\}$. The latter is trivially equivalent to the query $QE(F_{i,j})$ = t_{j_1} `OR ... OR` t_{j_u}.

In case $F_{i,j} = \neg\exists p \ hasPos(n, p) \wedge \bigwedge_t H_t(n, p)$, then $QE(F_{i,j})$ = `NOT` $QE(\neg F_{i,j})$, where $\neg F_{i,j}$ is transformed as in the previous case.

Theorem 5: Completeness of COMP

Proof Sketch: We will prove that every calculus query can be represented by a COMP query $CompQuery$. We use induction on the structure of the query expression $CalcExpr(n, p_1, ..., p_k)$.

- If $CalcExpr(n, p) = hasPos(n, p)$, then $CompQuery$ = p `HAS ANY`. This is equivalent to $CalcExpr$ by definition.

- If $CalcExpr(n, p) = hasToken(p, 'token')$, then $CompQuery$ = p `HAS 'token'`. This is equivalent to $CalcExpr$ by definition.

- If $CalcExpr(n, p_1, ..., p_k) = pred(p_1, ..., p_k, c_1, ..., c_q)$, then $CompQuery = \texttt{pred}(\texttt{p}_1, ..., \texttt{p}_k, \texttt{c}_1, ..., \texttt{c}_q)$. This is equivalent to $CalcExpr$ by definition.

- If $CalcExpr(n, p_1, ..., p_l, q'_1, ..., q'_m, q''_1, ..., q''_c)$ = $CalcExpr_1(n, p_1, ..., p_l, q'_1, ..., q'_m) \wedge$ $CalcExpr_2(n, p_1, ..., p_l, q''_1, ..., q''_c)$, where $k = l + m + c$, $CalcExpr_1$, and $CalcExpr_2$ are calculus query expressions with equivalent COMP queries be $CompQuery_1$ and $CompQuery_2$, then $CompQuery$ = $CompQuery_1$ AND $CompQuery_2$. This is equivalent to $CalcExpr$ by definition.

- If $CalcExpr(n, p_1, ..., p_l, q'_1, ..., q'_m, q''_1, ..., q''_c)$ = $CalcExpr_1(n, p_1, ..., p_l, q'_1, ..., q'_m) \vee$ $CalcExpr_2(n, p_1, ..., p_l, q''_1, ..., q''_c)$, where $k = l + m + c$, $CalcExpr_1$, and $CalcExpr_2$ are calculus query expressions with equivalent COMP queries be $CompQuery_1$ and $CompQuery_2$, then $CompQuery$ = $CompQuery_1$ OR $CompQuery_2$. This is equivalent to $CalcExpr$ by definition.

- If $CalcExpr(n, p_1, ..., p_k) = \neg CalcExpr'(n, p_1, ..., p_k)$, where $CalcExpr'$ is a calculus query expression that is equivalent to the COMP query $CompQuery'$, then $CompQuery$ = NOT $CompQuery'$. This is equivalent to $CalcExpr$ by definition.

- If $CalcExpr(n, p_1, ..., p_k)$ = $\exists p_{k+1}\ hasNode(n, p_{k+1}) \wedge CalcExpr'(n, p_1, ..., p_{k+1})$, where $CalcExpr'$ is a calculus query expression that is equivalent to the COMP query $CompQuery'$, then $CompQuery$ = SOME p_{k+1} ($CompQuery'$). This is equivalent to $CalcExpr$ by definition.

- If $CalcExpr(n, p_1, ..., p_k) = \forall p_{k+1}\ hasNode(n, p_{k+1}) \Rightarrow CalcExpr'(n, p_1, ..., p_{k+1})$, where $CalcExpr'$ is a calculus query expression that is equivalent to the COMP query $CompQuery'$, then $CompQuery$ = EVERY p_{k+1} ($CompQuery'$) . This is equivalent to $CalcExpr$ by definition. □